

# Communication-Avoiding Algorithms and Fast Matrix Multiplication

Grey Ballard

based on joint work with

Austin Benson, James Demmel, Benjamin Lipshitz, Oded Schwartz, and many others

September 29, 2014



**Sandia National Laboratories**

# Outline

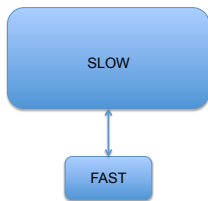
- 1 Communication Costs: Lower Bounds & Algorithms
- 2 Strassen's Matrix Multiplication: Theory & Practice
- 3 Searching for Fast Matrix Multiplication
- 4 Practical Performance of Fast Matrix Multiplication

# We must consider communication

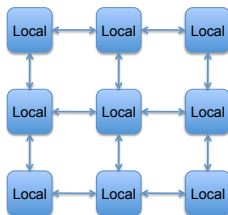
Algorithms have two kinds of costs: computation and *communication*

- moving data within memory hierarchy on a sequential computer
- moving data between processors on a parallel computer

For high-level analysis, we need simple memory models:



Sequential



Parallel

# Runtime Model

Measure computation in terms  
of # *flops* performed

Time per flop:  $\gamma$

Measure communication in terms  
of # *words* communicated

Time per word:  $\beta$

Total running time of an algorithm (ignoring overlap):

$$\gamma \cdot (\# \text{ flops}) + \beta \cdot (\# \text{ words})$$

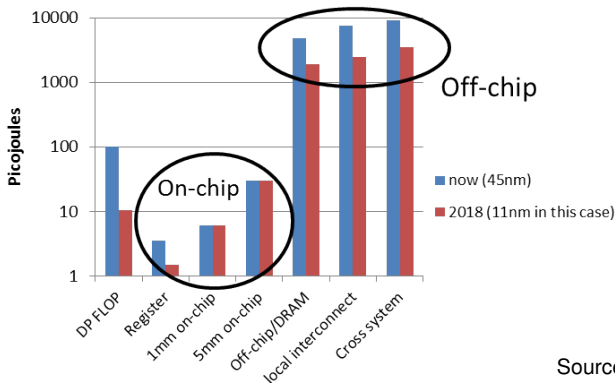
$\beta \gg \gamma$  as measured in time *and* energy, and the relative cost of communication is increasing

# Why avoid communication

## Annual Improvements in Time

Flop rate $\gamma$	DRAM Bandwidth $\beta$	Network Bandwidth $\beta$
59% per year	23% per year	26% per year

## Energy cost comparisons



Source: John Shalf

# Lower bounds and algorithms

Suppose you have a bottleneck in a computation you care about,  
how do you evaluate your options?

Performance models based on only computational complexity are no longer sufficient—we must analyze communication costs.

Communication lower bounds and optimal algorithms are known for some regular computations (e.g. matmul, FFT/sorting, SpMV, stencils), but irregular (more data-dependent) computations are harder.

# Lower bounds for classical matrix multiplication

- Assume  $\Theta(n^3)$  algorithm
- Sequential case with fast memory of size  $M$ 
  - lower bound on words moved between fast/slow mem:

$$\Omega\left(\frac{n^3}{\sqrt{M}}\right) \quad [\text{Hong \& Kung 81}]$$



- attained by blocked algorithm
- Parallel case with  $P$  processors (local memory of size  $M$ )
  - lower bound on words communicated (along critical path):

$$\Omega\left(\frac{n^3}{P\sqrt{M}}\right) \quad [\text{Toledo et al. 04}]$$



- also attainable

# Extensions to the rest of linear algebra

Theorem (B., Demmel, Holtz, Schwartz 11)

*If a computation “smells” like 3 nested loops, it must communicate*

$$\# \text{ words} = \Omega \left( \frac{\# \text{ flops}}{\sqrt{\text{memory size}}} \right)$$

This result applies to

- dense or sparse problems
- sequential or parallel computers

# Extensions to the rest of linear algebra

## Theorem (B., Demmel, Holtz, Schwartz 11)

*If a computation “smells” like 3 nested loops, it must communicate*

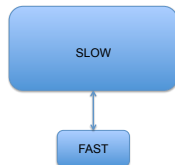
$$\# \text{ words} = \Omega \left( \frac{\# \text{ flops}}{\sqrt{\text{memory size}}} \right)$$

What smells like 3 nested loops?

- the rest of BLAS 3 (e.g. matrix multiplication, triangular solve)
- Cholesky, LU,  $LDL^T$ ,  $LTL^T$  decompositions
- QR decomposition
- eigenvalue and SVD reductions
- sequences of algorithms (e.g. repeated matrix squaring)
- graph algorithms (e.g. all pairs shortest paths)

# Optimal algorithms - sequential $\Theta(n^3)$ linear algebra

Computation	Optimal Algorithm
BLAS 3	blocked algorithms [Gustavson 97]
Cholesky	LAPACK [Ahmed & Pingali 00] <a href="#">[BDHS10]</a>
Symmetric Indefinite	LAPACK (rarely) <a href="#">[BBD<sup>+</sup>13]</a>
LU	LAPACK (rarely) [Toledo 97]* [Grigori et al. 11]
QR	LAPACK (rarely) [Frens & Wise 03] [Elmroth & Gustavson 98]* [Hoemmen et al. 12]*
Eig, SVD	<a href="#">[BDK12]</a> , <a href="#">[BDD11]</a>



# Example practical speedups

- Computing QR decomposition
  - up to **8× speedup** on multicore, **3× speedup** on GPU
- Solving symmetric indefinite linear systems
  - up to **3× speedup** on multicore
- Rectangular matrix multiplication (classical)
  - up to **7× speedup** on multicore
- Solving the symmetric eigenproblem for band matrices
  - up to **6× speedup** on multicore

# Outline

- 1 Communication Costs: Lower Bounds & Algorithms
- 2 Strassen's Matrix Multiplication: Theory & Practice**
- 3 Searching for Fast Matrix Multiplication
- 4 Practical Performance of Fast Matrix Multiplication

# Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for  $2 \times 2$  matrix multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

## Strassen's Algorithm

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$

$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$

$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$

$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

# Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for  $2 \times 2$  matrix multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

## Classical Algorithm

$$\begin{aligned} M_1 &= A_{11} \cdot B_{11} \\ M_2 &= A_{12} \cdot B_{21} \\ M_3 &= A_{11} \cdot B_{12} \\ M_4 &= A_{12} \cdot B_{22} \\ M_5 &= A_{21} \cdot B_{11} \\ M_6 &= A_{22} \cdot B_{21} \\ M_7 &= A_{21} \cdot B_{12} \\ M_8 &= A_{22} \cdot B_{22} \\ C_{11} &= M_1 + M_2 \\ C_{12} &= M_3 + M_4 \\ C_{21} &= M_5 + M_6 \\ C_{22} &= M_7 + M_8 \end{aligned}$$

## Strassen's Algorithm

$$\begin{aligned} M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\ M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\ M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\ M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\ C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

# Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for  $2 \times 2$  matrix multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

## Strassen's Algorithm

For  $n \times n$  matrices, we split into quadrants and use recursion

Flop count recurrence:

$$F(n) = 7 \cdot F(n/2) + O(n^2)$$

$$F(1) = 1$$

$$F(n) = O(n^{\log_2 7})$$

$$\log_2 7 \approx 2.81$$

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$

$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$

$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$

$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

# Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for  $2 \times 2$  matrix multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

## Strassen's Algorithm

For  $n \times n$  matrices, we split into quadrants and use recursion

Word count recurrence:

$$W(n) = 7 \cdot W(n/2) + O(n^2)$$

$$W(\sqrt{M}) = O(M)$$

$$W(n) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$$

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$

$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$

$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$

$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right) \quad O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$$

**Par**



Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References:

# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#),

# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



$$\Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right)$$

Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#),

# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} \frac{M}{P} \right) \quad \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right)$$

Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#),

# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} \frac{M}{P} \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right)$$

Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#), [\[BDH<sup>+</sup>12a\]](#),

# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} \frac{M}{P} \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right) \\ + \\ O \left( \frac{n^2}{P^2 / \log_2 7} \right)$$

Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#), [\[BDH<sup>+</sup>12a\]](#),



# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} \frac{M}{P} \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right)$$

$$+ \quad +$$

$$\Theta \left( \frac{n^2}{P^2 / \log_2 8} \right) \quad \Theta \left( \frac{n^2}{P^2 / \log_2 7} \right)$$

Units = (max) words communicated

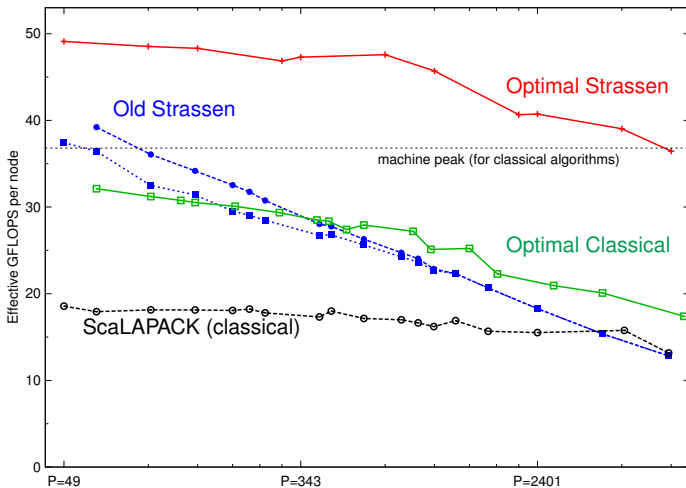
$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#), [\[BDH<sup>+</sup>12a\]](#), [\[BDH<sup>+</sup>12b\]](#),

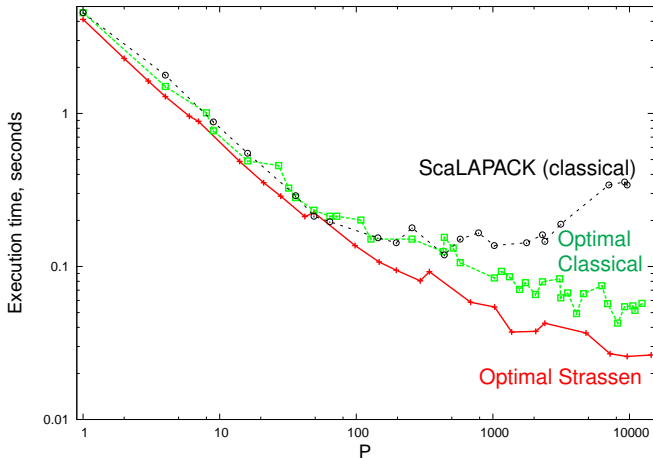
# Performance of optimal algorithms on large problem

Strong-scaling on a Cray XT4,  $n = 94,080$



# Execution time of optimal algorithms on small problem

Strong-scaling on a Cray XE6,  $n = 4704$



# Communication costs of matrix multiplication

**Classical**

**Strassen's**

**Seq**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

**Par**



$$\Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} \frac{M}{P} \right) \quad \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right)$$

$$+ \quad +$$

$$\Theta \left( \frac{n^2}{P^2 / \log_2 8} \right) \quad \Theta \left( \frac{n^2}{P^2 / \log_2 7} \right)$$



Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#), [\[BDH<sup>+</sup>12a\]](#), [\[BDH<sup>+</sup>12b\]](#),

# Communication costs of matrix multiplication

	Classical	Strassen's	Fast: $\Theta(n^{\omega_0})$ flops
<b>Seq</b> 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} M\right)$
<b>Par</b> 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$ $+$ $\Theta\left(\frac{n^2}{P^2/\log_2 8}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$ $+$ $\Theta\left(\frac{n^2}{P^2/\log_2 7}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \frac{M}{P}\right)$ $+$ $\Theta\left(\frac{n^2}{P^2/\omega_0}\right)$

Units = (max) words communicated

$O$  = algorithm exists,  $\Omega$  = lower bound exists,  $\Theta$  = both exist

$n$  = matrix dimension,  $M$  = fast/local memory size,  $P$  = number of processors

References: [\[BDHS11\]](#), [\[BDH<sup>+</sup>12a\]](#), [\[BDH<sup>+</sup>12b\]](#), [\[BDHS12\]](#), [\[BDHS14\]](#)

# How small can $\omega_0$ get?

People have worked on this problem for decades!

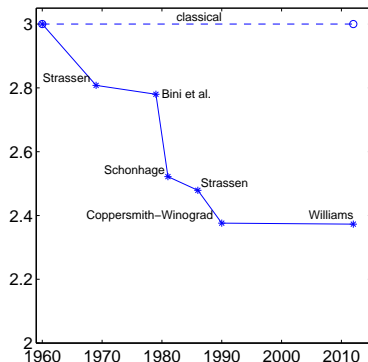
- “fast” algorithms multiply matrices using  $O(n^{\omega_0})$  flops,  $\omega_0 < 3$

# How small can $\omega_0$ get?

People have worked on this problem for decades!

- “fast” algorithms multiply matrices using  $O(n^{\omega_0})$  flops,  $\omega_0 < 3$

Exponent over time

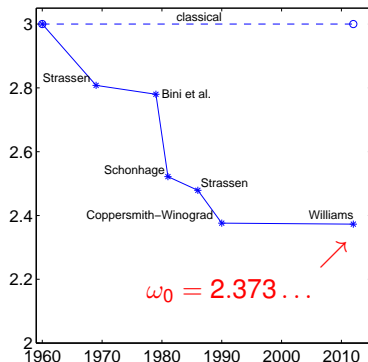


# How small can $\omega_0$ get?

People have worked on this problem for decades!

- “fast” algorithms multiply matrices using  $O(n^{\omega_0})$  flops,  $\omega_0 < 3$

Exponent over time



# How small can $\omega_0$ get?

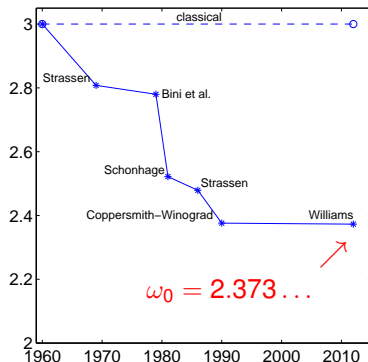
People have worked on this problem for decades!

- “fast” algorithms multiply matrices using  $O(n^{\omega_0})$  flops,  $\omega_0 < 3$

Most fast algorithms are only theoretical because they

- involve approximations
  - $A \cdot B = C + \lambda E$
- are not explicit
  - only proofs of existence
- have (possibly) large constants or log factors
  - most theoreticians care about only the exponent  $\omega$  in  $O(n^{\omega+\epsilon})$

Exponent over time



# Outline

- 1 Communication Costs: Lower Bounds & Algorithms
- 2 Strassen's Matrix Multiplication: Theory & Practice
- 3 Searching for Fast Matrix Multiplication**
- 4 Practical Performance of Fast Matrix Multiplication

# Practical Fast Algorithms

- Strassen's algorithm *is* practical
- Many algorithms are better in theory, are any better in practice?
- Can we find practical algorithms that have been overlooked?
- Can we implement and benchmark all known algorithms?

# Fast algorithms are based on recursion

Strassen showed how to use 7 multiplies instead of 8 for  $2 \times 2$  multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

## Classical Algorithm

$$\begin{aligned} M_1 &= A_{11} \cdot B_{11} \\ M_2 &= A_{12} \cdot B_{21} \\ M_3 &= A_{11} \cdot B_{12} \\ M_4 &= A_{12} \cdot B_{22} \\ M_5 &= A_{21} \cdot B_{11} \\ M_6 &= A_{22} \cdot B_{21} \\ M_7 &= A_{21} \cdot B_{12} \\ M_8 &= A_{22} \cdot B_{22} \\ C_{11} &= M_1 + M_2 \\ C_{12} &= M_3 + M_4 \\ C_{21} &= M_5 + M_6 \\ C_{22} &= M_7 + M_8 \end{aligned}$$

## Strassen's Algorithm

$$\begin{aligned} M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\ M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\ M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\ M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\ C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

# Recursion allows us to focus on base case

$$2 \times 2 \times 2$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

multiplies	6	7	8
flop count	$O(n^{2.58})$	$O(n^{2.81})$	$O(n^3)$

# Recursion allows us to focus on base case

$$2 \times 2 \times 2$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

multiplies	<del>6</del>	7	8
flop count	<del><math>O(n^{2.58})</math></del>	$O(n^{2.81})$	$O(n^3)$

# Recursion allows us to focus on base case

$$2 \times 2 \times 2$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

multiplies	<del>6</del>	7	8
flop count	<del><math>O(n^{2.58})</math></del>	$O(n^{2.81})$	$O(n^3)$

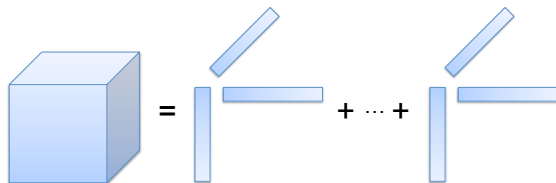
$$3 \times 3 \times 3$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

multiplies	19	21	23	27
flop count	$O(n^{2.68})$	$O(n^{2.77})$	$O(n^{2.85})$	$O(n^3)$

# Searching for a base case algorithm

Finding a better base case corresponds to computing a low-rank decomposition of a particular 3D tensor



$$\mathcal{T} = \sum_{r=1}^R \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

This is the main problem to solve

- various ways to attack it, but basically a search problem
- as base case gets bigger, tensor dimensions and rank get bigger

# Matrix multiplication as a tensor operation

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \mathbf{C}$$

is equivalent to

$$\mathcal{T} \times_1 \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{pmatrix} \times_2 \begin{pmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{pmatrix}$$


The diagram shows a 3D blue cube representing a 4x4x4 tensor  $\mathcal{T}$ . To its right is a vertical blue bar representing a 4x1 vector, with a  $\times_1$  label between them. Further right is a horizontal blue bar representing a 1x4 vector, with a  $\times_2$  label between them. An equals sign follows, and then a diagonal blue bar representing the resulting 4x1 vector.

where  $\mathcal{T}$  is a  $4 \times 4 \times 4$  tensor with the following slices:

$$\tau_1 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{pmatrix} \quad \tau_2 = \begin{pmatrix} & 1 & & \\ & & 1 & \\ & & & \\ & & & \end{pmatrix} \quad \tau_3 = \begin{pmatrix} & & 1 & \\ & & & 1 \\ & & & \\ & & & \end{pmatrix} \quad \tau_4 = \begin{pmatrix} & & & 1 \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

# Low-rank decomposition for Strassen

$$\mathcal{T} = \sum_{r=1}^7 \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

Strassen's decomposition is represented by these 3 factor matrices:

$$\mathbf{u} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$\mathbf{v} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{w} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

# Connection between factor matrices and algorithm

## Strassen's algorithm

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$

$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$

$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$

$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

## Strassen's factor matrices:

$$U = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$V = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

**U, V, W** matrices encode the algorithm

# Connection between factor matrices and algorithm

## Strassen's algorithm

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$

$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$

$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$

$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

$$C_{12} = M_3 + M_5$$

$$C_{21} = M_2 + M_4$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$

## Strassen's factor matrices:

		$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
<b>U</b>	$A_{11}$	1		1		1	-1	
	$A_{12}$					1		1
	$A_{21}$		1				1	
	$A_{22}$	1	1		1			-1
<b>V</b>	$B_{11}$	1	1		-1		1	
	$B_{12}$			1			1	
	$B_{21}$				1			1
	$B_{22}$	1		-1		1		1
<b>W</b>	$C_{11}$	1			1	-1		1
	$C_{12}$			1		1		
	$C_{21}$		1		1			
	$C_{22}$	1	-1	1			1	

**U, V, W** matrices encode the algorithm

# Main search problem

Given base case dimensions  $M$ ,  $P$ , and  $N$  (multiplying  $M \times P$  and  $P \times N$  matrices), the tensor  $\mathcal{T} \in \{0, 1\}^{MP \times PN \times MN}$  is specified.

# Main search problem

Given base case dimensions  $M$ ,  $P$ , and  $N$  (multiplying  $M \times P$  and  $P \times N$  matrices), the tensor  $\mathcal{T} \in \{0, 1\}^{MP \times PN \times MN}$  is specified.

Then for some desired rank  $R < MNP$ , find

$$\mathbf{U} \in \mathbb{F}^{MP \times R}, \mathbf{V} \in \mathbb{F}^{PN \times R}, \mathbf{W} \in \mathbb{F}^{MN \times R}$$

such that

$$t_{ijk} = \sum_{r=1}^R u_{ir} v_{jr} w_{kr} \quad \text{for all } i, j, k$$

(these  $(MNP)^2$  scalar constraints are equivalent to  $\mathcal{T} = \sum \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$ )

# Main search problem

Given base case dimensions  $M$ ,  $P$ , and  $N$  (multiplying  $M \times P$  and  $P \times N$  matrices), the tensor  $\mathcal{T} \in \{0, 1\}^{MP \times PN \times MN}$  is specified.

Then for some desired rank  $R < MNP$ , find

$$\mathbf{U} \in \mathbb{F}^{MP \times R}, \mathbf{V} \in \mathbb{F}^{PN \times R}, \mathbf{W} \in \mathbb{F}^{MN \times R}$$

such that

$$t_{ijk} = \sum_{r=1}^R u_{ir} v_{jr} w_{kr} \quad \text{for all } i, j, k$$

(these  $(MNP)^2$  scalar constraints are equivalent to  $\mathcal{T} = \sum \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$ )

- solution corresponds to algorithm with  $\omega_0 = 3 \log_{MPN} R$

# How do you solve it?

**Problem:** Find  $\mathbf{U}$ ,  $\mathbf{V}$ ,  $\mathbf{W}$  such that  $\mathcal{T} = \sum \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$

- the problem is NP-complete (for general  $\mathcal{T}$ )
- many combinatorial formulations of the problem
- efficient numerical methods can compute low-rank approximations
  - typical approach is “alternating least squares” (ALS)
  - pitfall: getting stuck at local minima  $> 0$
  - pitfall: facing ill-conditioned linear least squares problems
  - pitfall: numerical solution is good only to machine precision
- we seek exact, discrete, and sparse solutions

# Alternating least squares with regularization

Most successful scheme due to Smirnov [Smi13]

**Repeat**

1

$$\mathbf{U} = \arg \min_{\mathbf{U}} \left\| \mathbf{T}_{(U)} - \mathbf{U}(\mathbf{W} \odot \mathbf{V})^T \right\|_F^2 + \lambda \left\| \mathbf{U} - \tilde{\mathbf{U}} \right\|_F^2$$

2

$$\mathbf{V} = \arg \min_{\mathbf{V}} \left\| \mathbf{T}_{(V)} - \mathbf{V}(\mathbf{W} \odot \mathbf{U})^T \right\|_F^2 + \lambda \left\| \mathbf{V} - \tilde{\mathbf{V}} \right\|_F^2$$

3

$$\mathbf{W} = \arg \min_{\mathbf{W}} \left\| \mathbf{T}_{(W)} - \mathbf{W}(\mathbf{V} \odot \mathbf{U})^T \right\|_F^2 + \lambda \left\| \mathbf{W} - \tilde{\mathbf{W}} \right\|_F^2$$

**Until convergence**

Art of optimization scheme in tinkering with  $\lambda$ ,  $\tilde{\mathbf{U}}$ ,  $\tilde{\mathbf{V}}$ ,  $\tilde{\mathbf{W}}$  (each iteration)

# Discovered algorithms

Algorithm base case	Multiplies (fast)	Multiplies (classical)	Speedup per recursive step	$\omega_0$
$\langle 2, 2, 3 \rangle$	11	12	9%	2.89
$\langle 2, 2, 5 \rangle$	18	20	11%	2.89
$\langle 2, 2, 2 \rangle$ [Str69]	7	8	14%	2.81
$\langle 2, 2, 4 \rangle$	14	16	14%	2.85
$\langle 3, 3, 3 \rangle$	23	27	17%	2.85
$\langle 2, 3, 3 \rangle$	15	18	20%	2.81
$\langle 2, 3, 4 \rangle$	20	24	20%	2.83
$\langle 2, 4, 4 \rangle$	26	32	23%	2.82
$\langle 3, 3, 4 \rangle$	29	36	24%	2.82
$\langle 3, 4, 4 \rangle$	38	48	26%	2.82
$\langle 3, 3, 6 \rangle$ [Smi13]	40	54	35%	2.77
$\langle 2, 2, 3 \rangle^*$ [BCRL79]	10	12	20%	2.78
$\langle 3, 3, 3 \rangle^*$ [Sch81]	21	27	29%	2.77

## Example algorithm: $\langle 4, 2, 4 \rangle$

Partition matrices like this:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix}$$

- 1 Take 26 linear combos of  $A_{ij}$ 's according to  $\mathbf{U}$  (68 adds)
- 2 Take 26 linear combos of  $B_{ij}$ 's according to  $\mathbf{V}$  (52 adds)
- 3 Perform 26 multiplies (recursively)
- 4 Take linear combos of outputs to form  $C_{ij}$ 's acc. to  $\mathbf{W}$  (69 adds)

Classical algorithm performs 32 multiplies yielding a possible speedup of 23% per step

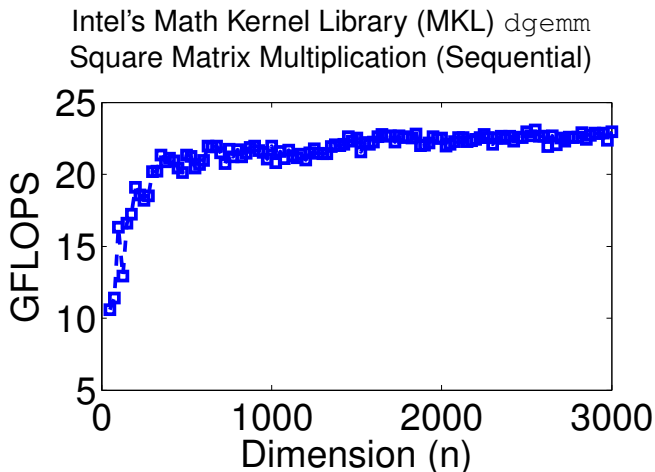
# Outline

- 1 Communication Costs: Lower Bounds & Algorithms
- 2 Strassen's Matrix Multiplication: Theory & Practice
- 3 Searching for Fast Matrix Multiplication
- 4 Practical Performance of Fast Matrix Multiplication**

# How do these algorithms perform in practice?

- All these algorithms have the same structure:
  - perform additions according to **U**, **V**, **W**, and make recursive calls
- Code generator can translate **U**, **V**, **W** into an implementation
- Sequential performance is based on:
  - classical multiplication implementation performance (vendor library)
  - efficiency of additions
  - crossover point of fast to classical
- Parallel performance depends also on parallelization approach

# Classical performance



- shape of  $\text{dgemm}$  curve gives rule of thumb for crossover point

# Performing (and optimizing) additions

Additions are completely memory bandwidth bound

- time is proportional to communication (flops are free)

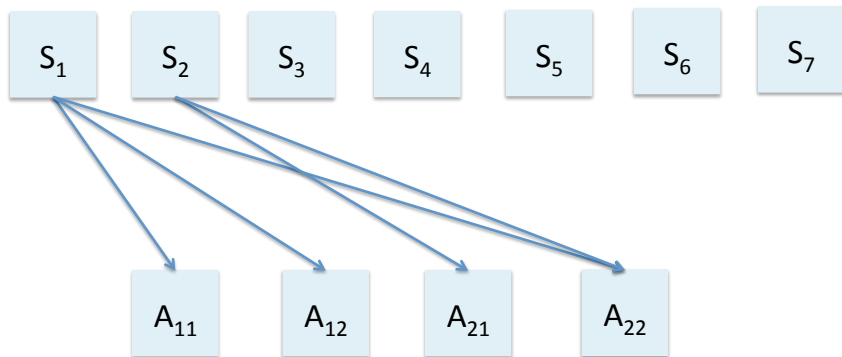
We micro-benchmarked three approaches:

- 1 Pairwise: most straightforward
- 2 Streaming: minimizes communication (in theory)
- 3 Write-once: best performance

We also considered common subexpression elimination

- 1 can help pairwise and streaming approaches
- 2 often hurts write-once approach

# Write-once approach to additions

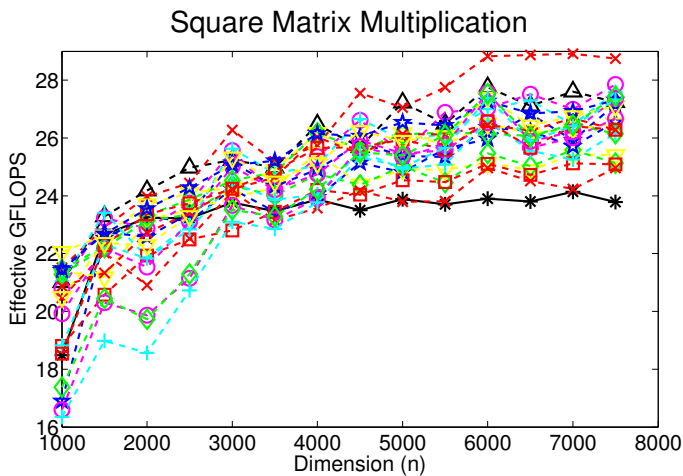


$$S_1 = A_{11} - A_{12} \quad + A_{22}$$

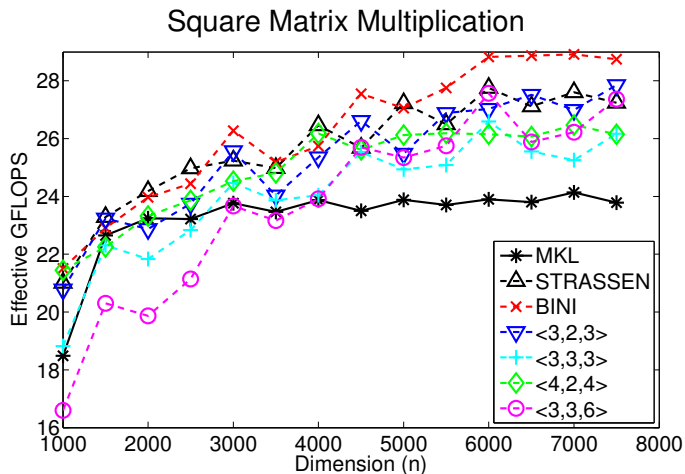
$$S_2 = \quad \quad \quad A_{21} - A_{22}$$

$$\vdots$$

# Sequential performance of fast algorithms

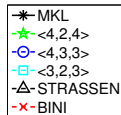
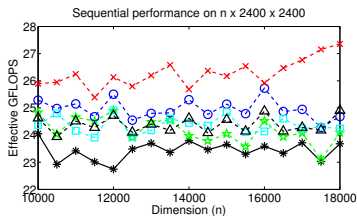
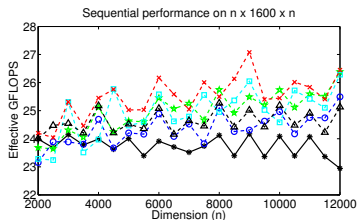


# Sequential performance of fast algorithms



# Sequential performance of fast algorithms

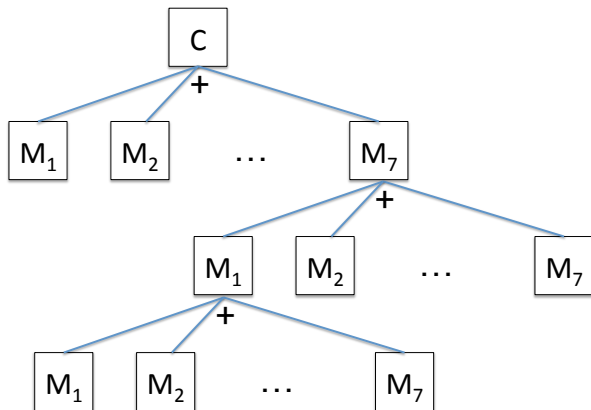
## Rectangular Matrix Multiplication



- best algorithms match “shape” of problem

# Parallelization schemes: recursion tree traversal

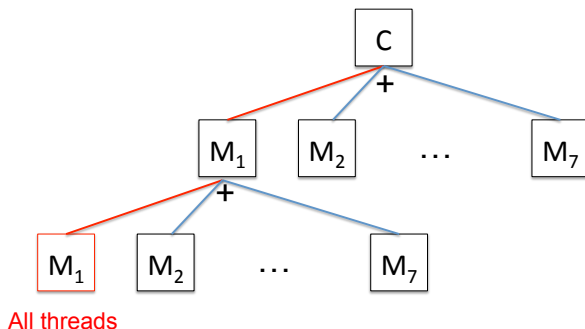
We consider 3 methods for shared-memory parallelization,  
based on traversing recursion tree



# Parallelization schemes: recursion tree traversal

**DFS:** depth first search is simplest scheme

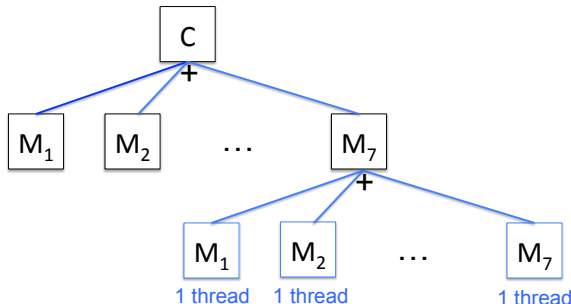
- all parallelism in calls to `dgemm`, always load balanced
- requires large subproblems for high performance



# Parallelization schemes: recursion tree traversal

**BFS:** breadth first search relies on sequential `dgemm`

- maintains high performance for small subproblems
- load balancing of multiplies is no longer guaranteed

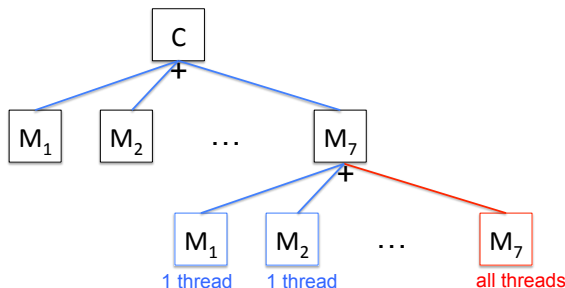


- 2 steps of Strassen creates 49 subproblems; we have 24 cores

# Parallelization schemes: recursion tree traversal

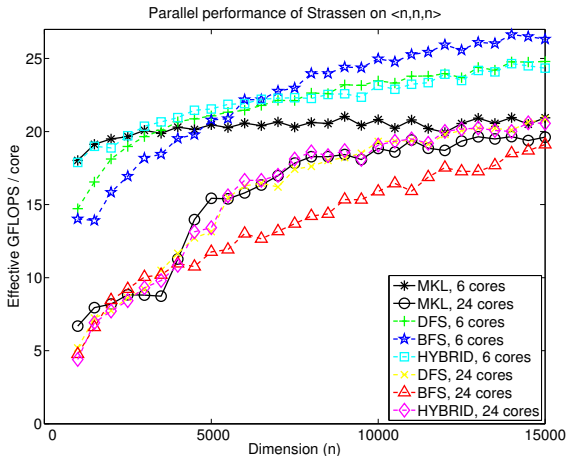
## HYBRID

- use BFS as much as possible
- use DFS to load balance leftovers

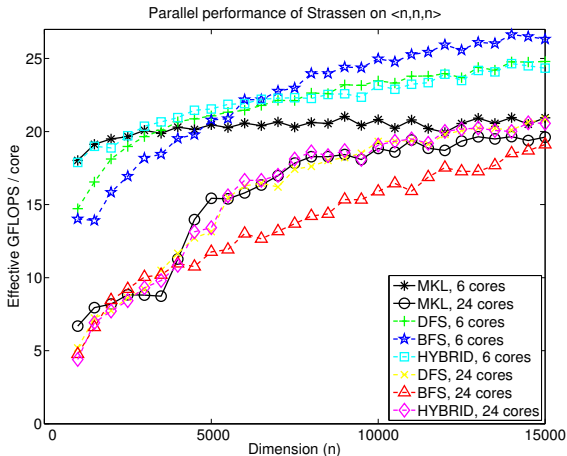


- 2 steps of Strassen creates 49 subproblems; we have 24 cores

# Parallel performance of fast algorithms



# Parallel performance of fast algorithms



- at 24 threads, not only are the additions bandwidth bound, but they don't scale as well as the multiplies (bandwidth scaling is  $< 6\times$ )

► Rectangular shapes

# Conclusions

- in theory, fast algorithms reduce both computation and communication
- in practice, fast algorithms like Strassen's can outperform `dgemm`
- for square matrices, Strassen's algorithm is hard to beat
- for rectangular matrices, algorithm should match the shape
- shared-memory parallelization faces bandwidth bottleneck

# Open questions

- 1 What are the numerical properties of all these algorithms?
- 2 How will they perform on distributed-memory parallel architectures?
- 3 Have we exhausted the possibilities of practical fast algorithms?
- 4 Can we use fast algorithms in the context of linear algebra and other applications?

# Communication-Avoiding Algorithms and Fast Matrix Multiplication

Grey Ballard

# Thank You!

`gmballa@sandia.gov`  
`www.sandia.gov/~gmballa`

# Extra slides

- 1 ▶ Optimal Parallel Algorithms
- 2 ▶ Matmul-as-tensor-operation using low-rank decomposition
- 3 ▶ Classical algorithm's factor matrices
- 4 ▶ Bini's factor matrices
- 5 ▶ Code generator performance comparison
- 6 ▶ Parallel performance for rectangular shapes

# Algorithms - Parallel $\Theta(n^3)$ Linear Algebra

Algorithm	Reference	Factor exceeding lower bound for # words	Factor exceeding lower bound for # messages
Matrix Multiply	[Can69]	1	1
Cholesky	ScaLAPACK	$\log P$	$\log P$
Symmetric Indefinite	[BBD <sup>+</sup> 13] ScaLAPACK	? $\log P$	? $(N/P^{1/2}) \log P$
LU	[GDX11] ScaLAPACK	$\log P$ $\log P$	$\log P$ $(N/P^{1/2}) \log P$
QR	[DGHL12] ScaLAPACK	$\log P$ $\log P$	$\log^3 P$ $(N/P^{1/2}) \log P$
SymEig, SVD	[BDK12] ScaLAPACK	? $\log P$	? $N/P^{1/2}$
NonsymEig	[BDD11] ScaLAPACK	$\log P$ $P^{1/2} \log P$	$\log^3 P$ $N \log P$

\*This table assumes that *one* copy of the data is distributed evenly across processors

Red = not optimal



# Matmul-as-tensor-operation using low-rank decomposition

Here's the matrix multiplication as tensor operation again:

$$\mathcal{T} \times_1 \mathbf{a} \times_2 \mathbf{b} = \mathbf{c}$$

Here's our low-rank decomposition:

$$\mathcal{T} = \sum_{r=1}^R \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

Here's an encoding of our new matrix multiplication algorithm:

$$\mathcal{T} \times_1 \mathbf{a} \times_2 \mathbf{b} = \sum_{r=1}^R (\mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r) \times_1 \mathbf{a} \times_2 \mathbf{b} = \sum_{r=1}^R (\mathbf{a}^T \mathbf{u}_r) \cdot (\mathbf{b}^T \mathbf{v}_r) \cdot \mathbf{w}_r$$

# Connection between factor matrices and algorithm

Classical algorithm:

$$\begin{aligned}M_1 &= A_{11} \cdot B_{11} \\M_2 &= A_{12} \cdot B_{21} \\M_3 &= A_{11} \cdot B_{12} \\M_4 &= A_{12} \cdot B_{22} \\M_5 &= A_{21} \cdot B_{11} \\M_6 &= A_{22} \cdot B_{21} \\M_7 &= A_{21} \cdot B_{12} \\M_8 &= A_{22} \cdot B_{22} \\C_{11} &= M_1 + M_2 \\C_{12} &= M_3 + M_4 \\C_{21} &= M_5 + M_6 \\C_{22} &= M_7 + M_8\end{aligned}$$

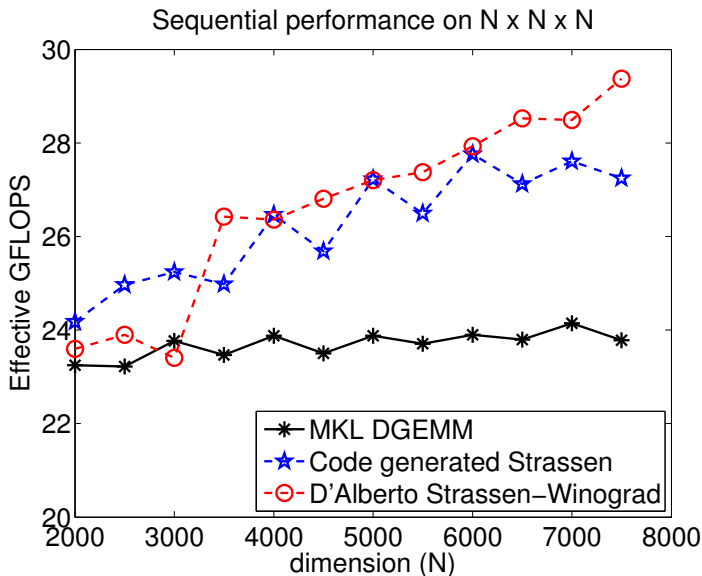
Classical factor matrices:

$$\mathbf{U} = \begin{pmatrix} 1 & & 1 & & & & \\ & 1 & & 1 & & & \\ & & & & 1 & & 1 \\ & & & & & 1 & \\ & & & & & & 1 \end{pmatrix}$$
$$\mathbf{V} = \begin{pmatrix} 1 & & & 1 & & & \\ & 1 & & & & 1 & \\ & & 1 & & 1 & & \\ & & & 1 & & & 1 \end{pmatrix}$$
$$\mathbf{W} = \begin{pmatrix} 1 & 1 & & & & & \\ & & 1 & 1 & & & \\ & & & & 1 & 1 & \\ & & & & & & 1 & 1 \end{pmatrix}$$

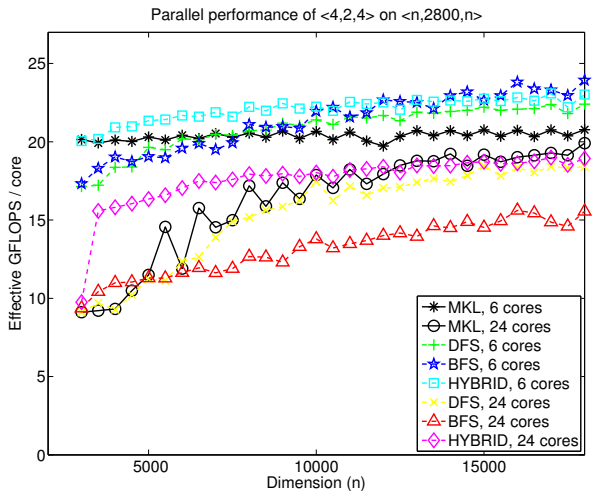
# Factor matrices for an approximate algorithm (Bini's)

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & \lambda \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$
$$\mathbf{V} = \begin{bmatrix} \lambda & 0 & 0 & -\lambda & 0 & 1 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 0 & 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & \lambda & 0 \\ 1 & -1 & 1 & 0 & 1 & \lambda & 0 & 0 & 0 & -\lambda \end{bmatrix}$$
$$\mathbf{W} = \begin{bmatrix} \frac{1}{\lambda} & \frac{1}{\lambda} & -\frac{1}{\lambda} & \frac{1}{\lambda} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\lambda} & 0 & \frac{1}{\lambda} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\lambda} & 0 & \frac{1}{\lambda} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\lambda} & -\frac{1}{\lambda} & \frac{1}{\lambda} & 0 & \frac{1}{\lambda} \end{bmatrix}$$

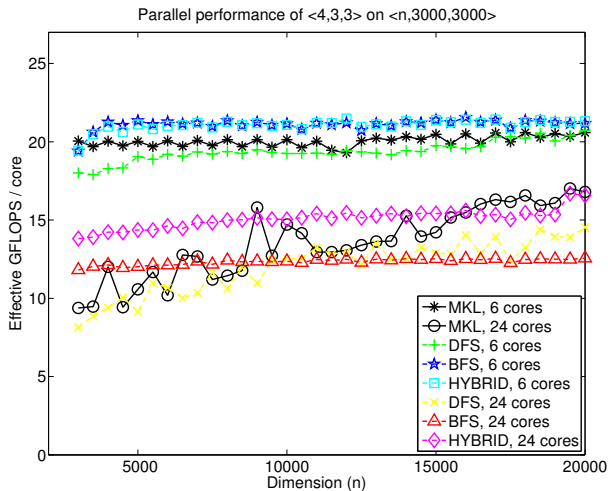
# Code generated vs tuned performance



# Parallel performance of fast algorithms



# Parallel performance of fast algorithms



# References I



G. Ballard, D. Becker, J. Demmel, J. Dongarra, A. Druinsky, I. Peled, O. Schwartz, S. Toledo, and I. Yamazaki.  
Communication-avoiding symmetric-indefinite factorization.  
Technical Report UCB/EECS-2013-127, EECS Department, University of California, Berkeley, July 2013.



D. Bini, M. Capovani, F. Romani, and G. Lotti.  
 $O(n^{2.7799})$  complexity for  $n \times n$  approximate matrix multiplication.  
*Information Processing Letters*, 8(5):234 – 235, 1979.



G. Ballard, J. Demmel, and I. Dumitriu.  
Communication-optimal parallel and sequential eigenvalue and singular value algorithms.  
Technical Report EECS-2011-14, UC Berkeley, February 2011.



G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz.  
Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds.  
In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 77–79, New York, NY, USA, 2012. ACM.



G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz.  
Communication-optimal parallel algorithm for Strassen's matrix multiplication.  
In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 193–204, New York, NY, USA, 2012. ACM.



G. Ballard, J. Demmel, O. Holtz, and O. Schwartz.  
Communication-optimal parallel and sequential Cholesky decomposition.  
*SIAM Journal on Scientific Computing*, 32(6):3495–3523, 2010.

# References II



G. Ballard, J. Demmel, O. Holtz, and O. Schwartz.

Graph expansion and communication costs of fast matrix multiplication.

In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 1–12. ACM, 2011.



G. Ballard, J. Demmel, O. Holtz, and O. Schwartz.

Graph expansion and communication costs of fast matrix multiplication.

*Journal of the ACM*, 59(6):32:1–32:23, December 2012.



Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz.

Communication costs of strassen's matrix multiplication.

*Commun. ACM*, 57(2):107–114, February 2014.



G. Ballard, J. Demmel, and N. Knight.

Communication avoiding successive band reduction.

In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPOPP '12, pages 35–44, New York, NY, USA, 2012. ACM.



L. Cannon.

*A cellular computer to implement the Kalman filter algorithm.*

PhD thesis, Montana State University, Bozeman, MN, 1969.



Paolo D'Alberto, Marco Bodrato, and Alexandru Nicolau.

Exploiting parallelism in matrix-computation kernels for symmetric multiprocessor systems: Matrix-multiplication and matrix-addition algorithm optimizations by software pipelining and threads allocation.

*ACM Trans. Math. Softw.*, 38(1):2:1–2:30, December 2011.

# References III



J. Demmel, L. Grigori, M. Hoemmen, and J. Langou.

Communication-optimal parallel and sequential QR and LU factorizations.  
*SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012.



L. Grigori, J. Demmel, and H. Xiang.

CALU: A communication optimal LU factorization algorithm.  
*SIAM Journal on Matrix Analysis and Applications*, 32(4):1317–1350, 2011.



A. Schönhage.

Partial and total matrix multiplication.  
*SIAM Journal on Computing*, 10(3):434–455, 1981.



A.V. Smirnov.

The bilinear complexity and practical algorithms for matrix multiplication.  
*Computational Mathematics and Mathematical Physics*, 53(12):1781–1795, 2013.



V. Strassen.

Gaussian elimination is not optimal.  
*Numerische Mathematik*, 13:354–356, 1969.  
10.1007/BF02165411.